Agenda

* Iteratively remove rows and columns that have a higher than threshold proportion of missing data
* Test combinations of different thresholds, imputation methods, and models on model prediction accuracy.

**Iteratively thresholding**:

For each threshold, I iteratively removed rows and columns with more missing data than the threshold until the difference between the current proportion of missing data and the proportion of missing data was less than 0.005, or 0.5%.

The proportion of missing data left in the dataset after thresholding increased steadily with the threshold value.

A graph with blue dots

AI-generated content may be incorrect.

The number of rows remaining in the final dataset also increased with the threshold value. However, they increased less smoothly. There appeared to be two ‘jumps’ in the proportion of missing data (between 60 to 65% and 75 to 80% thresholds).

A graph with blue dots

AI-generated content may be incorrect.

The number of columns remaining in the final dataset increased as the threshold increased, but in this case the number of thresholds only began notably increasing for thresholds 90% and above.

A graph with blue dots

AI-generated content may be incorrect.

**Model Testing Methodology**:

1. Each thresholded dataset was split into training/testing sets (75:25% split), where data from the same year could be in the training or testing set.
2. Data imputation was performed on a copy of each thresholded training/testing set. These imputation methods included:
   1. K-Nearest Neighbours
   2. Miss Forest
   3. Polynomial order 1 (linear)
   4. Polynomial order 2 (quadratic)
   5. Polynomial order 3 (cubic)
3. I tested the following models with each imputed version of each thresholded dataset:
   1. Random Forest
   2. Linear Regression
   3. Support Vector Machine Regression
   4. LightGBM
   5. XGBoost
   6. AdaBoost
      1. An ensemble method that trains successive weighted decision trees that weight examples that were errors in the previous iteration of the model more highly.
   7. Gaussian Process

To confirm the methodology, the imputation was performed on the testing and training datasets separately.

*Model Accuracy Metric:*

To test the accuracy of each model, I use MAPE (mean average percentage error).

Random Forest:

A graph with numbers and dots

AI-generated content may be incorrect.

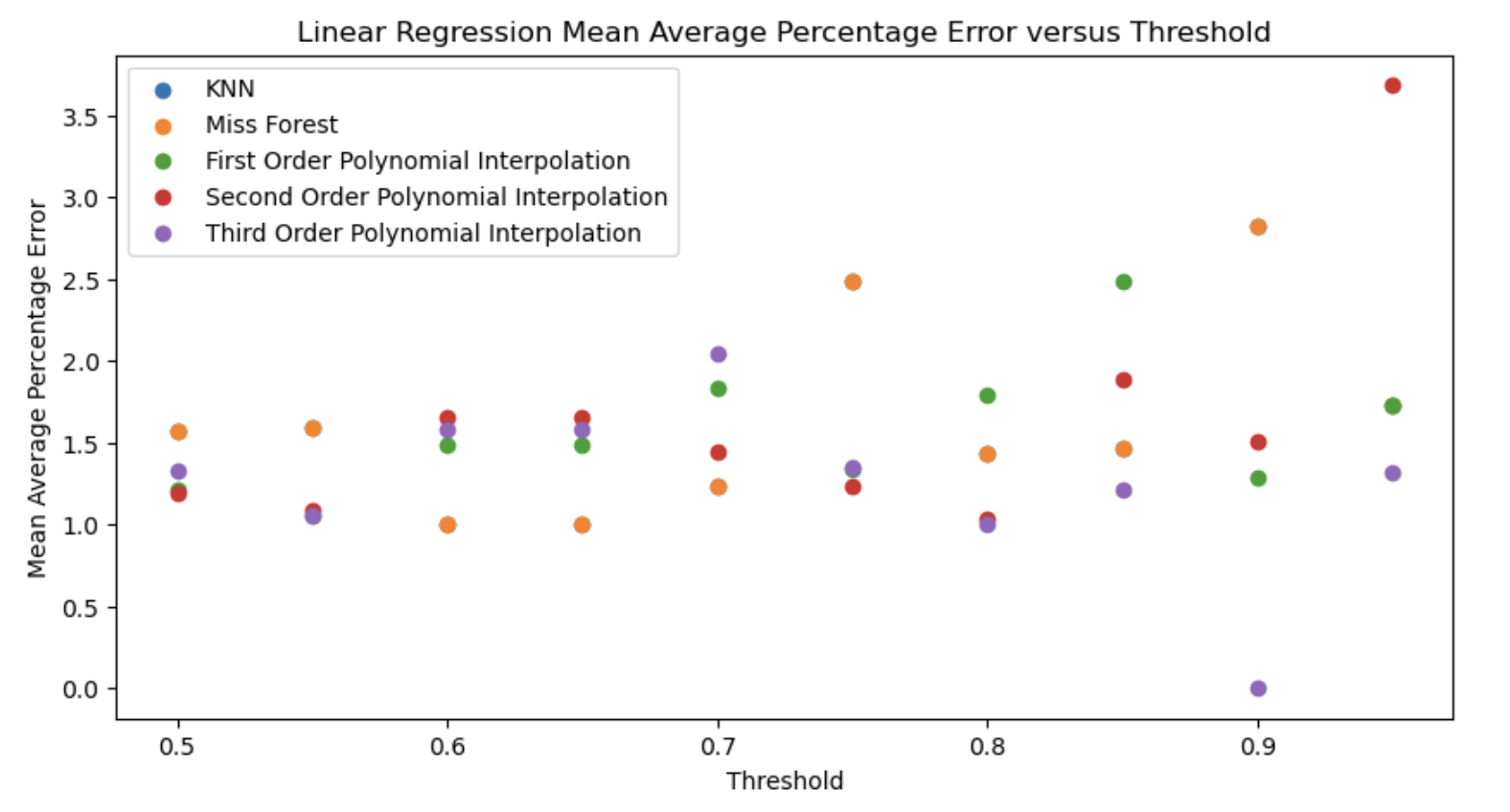
* The random forest model has the lowest MAPE when it is used on a dataset with a 55% threshold and first order polynomial imputation.
* It performs the worst on Miss Forest imputation.
* Its MAPE score ranges between 0.1 and 0.2

Linear Regression

A graph with numbers and dots

AI-generated content may be incorrect.

* There was a large outlier when testing the linear regression model with a 90% threshold and 3rd order polynomial imputation. This outlier’s value was set as zero in the re-displayed version of the plot below.



* The best MAPE scores for the linear regression model were between 1.0 and 1.5, representing error between 100 and 150% of the original value.
* While the different imputation methods performed similarly, Miss Forest and the 3rd order polynomial appeared to perform the best.
* The best thresholds appeared to be 65% and 80%, but there did not seem to be large differences.

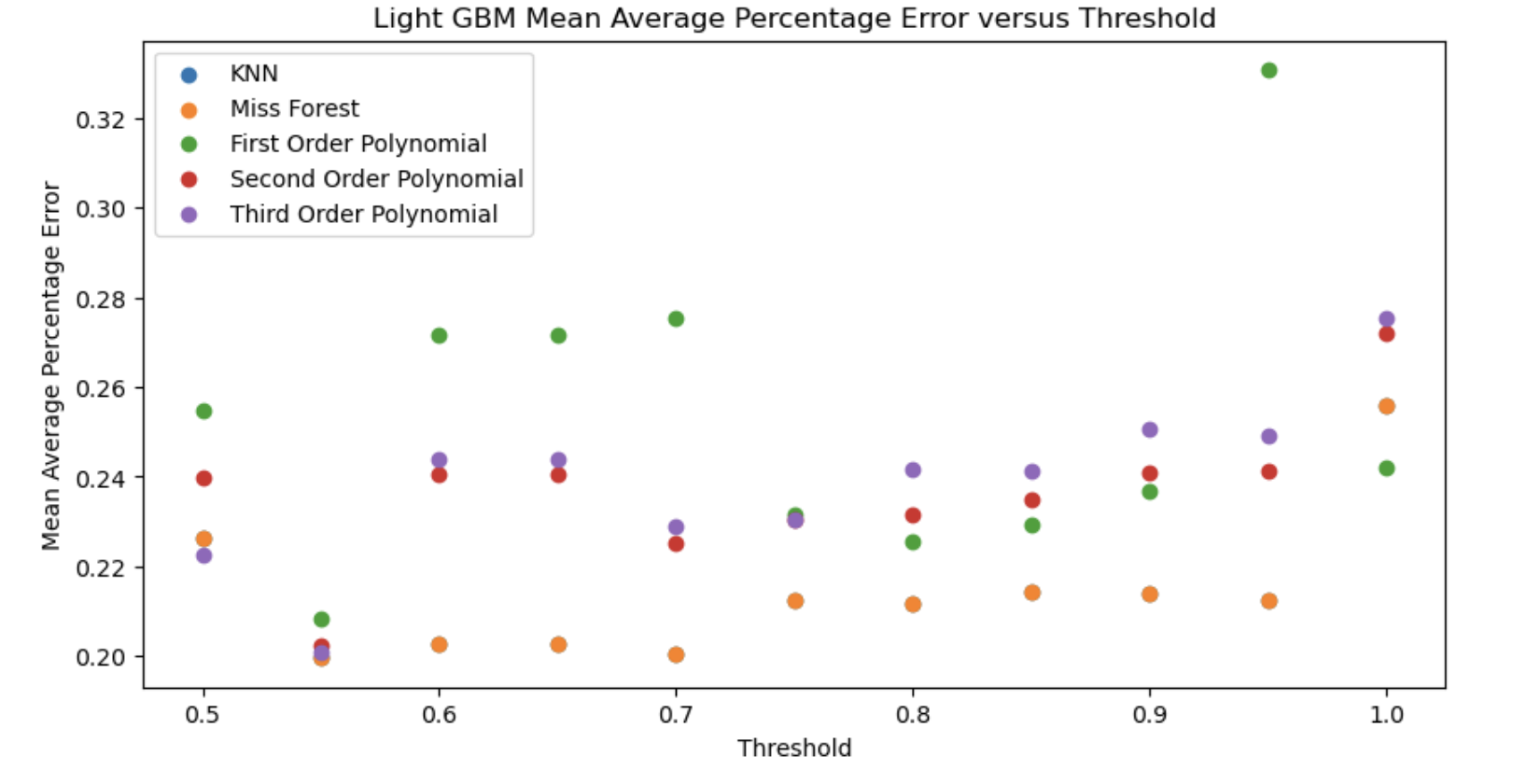
Support Vector Machine Regression:

A graph with colored dots and numbers

AI-generated content may be incorrect.

* A threshold of 80% with imputation using a 3rd order polynomial appeared to perform the best, with a larger range of MAPE scores than seen in previous models (e.g. Random Forest).
* The best MAPE scores were just over 0.8.

LightGBM:



* The MAPE scores for LightGBM were similar to those of Random Forest, being between 0.2 and 0.3.
* Miss Forest imputation appeared to perform the best, with the most effective thresholds being 55 and 70%.

XGBoost:

A graph with colored dots and numbers

AI-generated content may be incorrect.

* As with the linear regression model, XGBoost had a large outlier that I set to zero and replotted.

A graph with colored dots and white text

AI-generated content may be incorrect.

* XGBoost’s MAPE generally ranged between 0.1 and 0.4, with thresholds of 60-70% appearing to perform the best.
* Miss Forest and 1st order polynomial imputation appeared to be the most effective. Miss Forest will be considered, as it is most effective over the highest performing thresholds.

AdaBoost:

A graph with colored dots and white text

AI-generated content may be incorrect.

* The different imputation methods appeared to have negligible effects on AdaBoost’s accuracy, as it’s MAPE score ranged between 0.53 and 0.59.
* The best performing combination was a threshold of 55% and imputation using a 1st order polynomial.
* Otherwise, the 2nd order polynomial generally appeared to be the most effective.
* AdaBoost’s best MAPE scores were worse than the best MAPE scores of the other gradient boosting models (LightGBM and XGBoost).

Gaussian Processes:

A graph with purple dots

AI-generated content may be incorrect.

* The Gaussian Process model unfortunately performed extremely poorly, with errors in the magnitude of 10^16.
  + After excluding thresholds above 0.8, I re-ran the model. Unfortunately, there were still errors that had high orders of magnitude (>10^10), indicating that the above results were not just outliers.
* This model performed the worst of any model tested thus far.

**Overall Reflections**:

* The best imputation method and threshold differed by model.
* As shown below, the best overall combination appeared to be the Random Forest with imputation by a 1st order polynomial.
* ‘Tying’ for second and third were LGBM and XGBoost with Miss Forest and 1st order polynomial imputation respectively.
* There did not appear to be a large threshold-dependent different in performance. However, the best performing model combination appeared to do slightly between with lower thresholds (especially 55%).

A graph with colored dots

AI-generated content may be incorrect.

Difficulties Splitting the Data **By** Year:

I had initially wanted to perform the same analysis for training/testing data split **by** year. However, I ran into two difficulties:

* If I thresholded the data before splitting by year, columns that had passed the threshold had the potential to only contain NAN values, preventing many of these models from being used.
* If I split the data into training and testing sets before performing thresholding, the training and testing set could end up with different columns removed, making them uncomparable.

How would you recommend approaching this problem?

**Potential Future Avenues:**

* Experimentation with hyperparameters.

Two sections:

* No filtering: threshold = 100%
  + Within this, compare different machine learning models
* Show results of threshold of 90%
  + Then compare different methods
  + Produce list of removed columns

Try pulling in everything

* Make list of all variables used